Verifiable Multi-Party Business Process Automation

Joosep Simm, Jamie Steiner, and Ahto Truu

Guardtime, A. H. Tammsaare tee 60, 11316 Tallinn, Estonia
{joosep.simm,jamie.steiner,ahto.truu}@guardtime.com

Abstract. Lack of trust is one of the main problems preventing multi-party business process automation. Solutions based on smart contracts and distributed ledgers have been proposed, but suffer from scalability issues. We present a more performant alternative approach to enable trust between business partners, based on authenticated data structures.

Keywords: Multi-party business process automation, distributed ledger technology, authenticated data structures, sparse Merkle trees.

1 Introduction

Business process automation (BPA) is the use of technology to execute recurring tasks or processes with the goal of replacing manual effort. Sometimes also referred to as digital transformation, its potential benefits include increasing service quality, improving service delivery, and reducing costs.

Most BPA deployments aim to automate a firm's internal operations. However, many business processes are composed of a series of steps taken by different firms. Single-party business processes such as invoice production and processing, which might be handled using a corporate enterprise resource planning (ERP) system, can be viewed as steps in the overarching multi-party business process of two or more firms engaging in a trade of goods or services. Thus, if BPA aims to improve performance of a firm's internal business activities, we may define multi-party business process automation (MPBPA) as the use of technology to optimize and automate firms' interactions with each other.

As an example, we may consider a hypothetical supply chain relationship. A Supplier regularly produces and ships widgets to a Customer. The widgets are transported by a series of third-party Logistics companies. Payment is typical net 30 terms. Since the Supplier is interested in being paid as soon as possible, he uses the services of a Bank, which pays the Supplier upon delivery to the first Logistics provider, in exchange for interest. This arrangement is common and is effectively a loan secured by the Supplier's receivables.

The challenges in realizing MPBPA differ from those solved by extant BPA technology. The most important difference is the inherent lack of trust between firms on matters of value. A firm would be naturally hesitant to allow a computer

program to automatically pay invoices; instead, review and approval by a trusted employee in accounts payable is typically required. If it were possible to automate such interactions in a trusted way, many of the same benefits that BPA has yielded within the scope of individual firms could be realized at an inter-company or even systemic level. In our example, the Customer receives the goods, and must reconcile the received amount against the invoice. Then they must make payment to the Bank according to their terms. The Bank receives payment and also reconciles it against the terms of the original invoice before extinguishing the loan to the Supplier.

The existence of current manual reconciliation processes shows that firms are willing to pay a substantial premium to ensure that business rules, designed to protect against fraud or error, are applied correctly.

Distributed ledger technology (DLT) can be described as the process of replicating and synchronizing shared data between several mutually distrusting parties. The data can then be relied on in situations where no single party can be trusted to create and maintain a central database. Byzantine fault tolerant (BFT) consensus protocols are typically used for the replication and synchronization, and are often seen as the underlying technological basis for DLT. These consensus algorithms, however, present a number of challenges.

First, BFT consensus has high network overhead. This has caused many practical implementations to appoint a small subset of nodes in the network as designated validators who receive all transactions. The validators then apply the business rules, which are encoded in the ledger itself as so-called smart contracts, to determine which transactions are valid, and follow the consensus protocol to agree on how the ledger state should be updated.

Second, clients might not want to share their confidential business data. In such cases, only the parties to a contract would receive the transaction details and know the code to be executed. In our example, trade information is commercially sensitive. For example, the supplier may not want other customers to know that they provide widgets to one customer at a discount, because other customers might also ask for a discounted price. Thus, the parties execute the code independently and only post attestations about the current state of the contract to the ledger. The validators follow the consensus protocol to ensure that the attestations are reliably recorded.

In the latter case, the validators do not know the transaction data or the contract code. Thus, they can not possibly know whether the contract has been executed correctly. The attestation of state is still useful, however, since all parties can be assured of having a common, non-repudiable view of the state. We refer to this arrangement as using "privacy limited validation." In a supply chain context, if the validators were also part of the supply chain, by validating their competitors' plain text transactions, they might be able to obtain commercially useful information that they would not otherwise be entitled to.

In summary, several key ingredients are necessary for multi-party business process automation:

– a process definition which describes the possible or acceptable interactions between the parties, i.e. the business rules;
– a reliable record of the information added by different parties, at different times, to contribute to and progress the execution of that process;
– means to prevent parties from repudiating the current process state, claiming an alternative process state, or claiming alternative histories of how the process progressed to the current state.

Smart contracts underpinned by DLT are one way to meet the above requirements. However, implementations that use them have proven difficult to scale. We propose an alternative approach that is more scalable and can be used as the basis for many of the multiparty process enforcement use cases that others address through DLT. Our solution is based on committing process states into an authenticated data structure (ADS) operated by a server whose actions can be independently verified.

2 Related work

Several DLT-based business process management (BPM) systems have been proposed to support execution of multi-party processes.

Caterpillar [7] is a BPM system that runs on top of the Ethereum blockchain. It keeps all process related data on the blockchain, in order to ensure its reliability. On one hand, no off-chain data is required to read or execute a process. On the other hand, potentially sensitive data must be posted to the blockchain.

Lorikeet [16] is a model-driven engineering tool. It generates smart contract code (in the Solidity language of the Ethereum blockchain) from a Business Process Model and Notation (BPMN) representation. It uses DLT for communication between parties, but sensitive data is not posted to the blockchain.

Both solutions suffer from the inherent limitations of DLT design, namely the overhead required to reach consensus between parties. This limits the throughput of these systems. A comparative overview of these systems is given in [5].

The Laava platform [17] focuses on multi-tenant aspects of BPM and proposes to solve these by creating a private blockchain instance for each tenant and periodically aggregating the states of these private chains into a commitment posted to a public blockchain as an external anchor. A drawback of this approach is that a user would still have to scan a whole private blockchain to extract, prove, or verify the history of one process.

There are other solutions outside of academic research, such as the Proof of Process by Stratumn [14]. Their system is also based on DLT, and therefore inherits its limitations.

Mendling et al. [8] discuss different directions in business processes and blockchain interaction. One of the proposed approaches is based on monitoring:

> This provides a suitable basis for continuous conformance and compliance checking and monitoring of service-level agreements. Second, based on monitoring data exchanged via the blockchain, it is possible to verify

if a process instance meets the original process model and the contractual obligations of all involved process stakeholders.

Our proposed solution is close to this description. It does not put restrictions on execution of business processes, but provides a secure, performant, and scalable way to monitor the execution of multi-party processes.

Breu et al. [2] introduced the distinction between process orchestration, where the process instance is managed by a central service provider, and process choreography, where the ownership of the instance is transferred from one participant to another. We note that our monitoring-based approach is applicable in both cases and perhaps even more valuable in the context of choreographies where it is easier for the parties to get confused over the current state of the process [11].

3 Preliminaries

A hash function h maps arbitrary-sized inputs to fixed-size outputs.

Hash values are often used as representatives of inputs that are either too large or too confidential to be handled directly. For example, in the hash-then-sign model, a document's hash value is signed instead of the document itself. Likewise, in the hash-then-publish time-stamping model, a hash value is published to establish evidence of the existence of the input without disclosing the input itself.

To facilitate such uses, cryptographic hash functions are required to have several additional properties, such as one-wayness (it is infeasible to reconstruct the input from the output), second pre-image resistance (it is infeasible to change the input so that it still maps to the same output), and collision resistance (it is infeasible to find two distinct inputs mapping to the same output). We refer to [12] for a more extensive treatment.

SHA-256 and SHA-512 are practical hash functions (with 256-bit and 512-bit outputs, respectively) that have all these properties and are expected to withstand cryptoanalytic attacks on them for at least the next ten years [10].

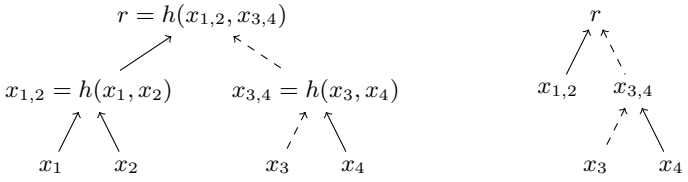

Fig. 1. Merkle tree with 4 leaves and the hash chain linking x_3 to r.

Hash trees [9], also known as Merkle trees (MT), are tree-shaped structures of hash values. Each node is either a leaf with no children or an internal node

with two children. The value x of an internal node is computed as $x = h(x_l, x_r)$, where x_l and x_r are the values of the left and right child, respectively. There is one root node r that is not a child of any node.

In order to prove that a value x_i participated in the computation of the root hash r, it is sufficient to present values of all the sibling nodes on the unique path from x_i to the root of the tree, along with the "shape" of the path (defining the concatenation order on each step). For example, to claim that x_3 belongs to the tree shown on the left in Fig. 1, one has to present the values x_4 and $x_{1,2}$ to enable the verifier to compute $x_{3,4} = h(x_3, x_4)$, $r = h(x_{1,2}, x_{3,4})$, essentially re-building a slice of the tree, as shown on the right in Fig. 1.

A standard hash tree can thus be used to prove that a value was in a set (so-called "inclusion proof"). It is, however, not suitable to proving that a value was absent from a set (so-called "exclusion proof"). The value could be in any node, and to verify its absence, the whole tree would have to be scanned.

A way to overcome this limitation is to dedicate a leaf to each potential value of the set. Then the leaves corresponding to values not in the set could be set to some "empty" sentinel value (shown as white in Fig. 2) and absence of a value in the set can be proven by showing that the dedicated leaf is empty.

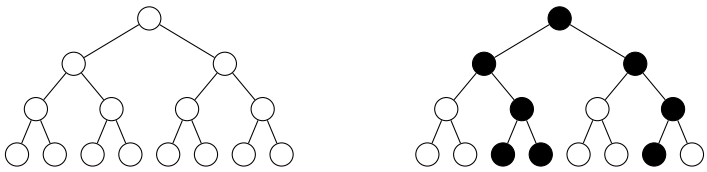

Fig. 2. Sparse Merkle trees: an empty one (left), and a populated one (right).

With 256-bit hash values, such a tree would have 2^{256} leaves, which is clearly infeasible. To resolve this, we amend the rule for populating internal nodes so that a parent of two empty child nodes also gets the "empty" value, and hashing is only used to compute values of parents with at least one non-empty child. Such a tree, called sparse Merkle tree (SMT), is fully defined by its non-empty nodes (shown as black in Fig. 2), the number of which is more tractable.

An SMT can also be used to manage maps of key-value pairs. In this case, the hash of the value is stored in the leaf dedicated to the hash of the key, and the leaves corresponding to absent keys will be left empty [1, 4].

4 Our approach

Verifiable business processes (VBP) is a transaction-based server-centric solution for adding trust to a registry or a service in order to achieve verifiable multi-party business process automation. In the case of traditional blockchains, new transactions are approved by majority agreement; VBP instead is based on single

trust domain where it generates proofs of correct operation, which would make incorrect behavior immediately evident. VBP consists of:

- authenticated data structures (Sec. 4.1);
- verifiable state machines (Sec. 4.2);
- verifiable log of registry changes (Sec. 4.3).

4.1 Authenticated Data Structures

An authenticated data structure (ADS) is a data structure whose operations can be carried out by an untrusted service provider, the results of which a verifier can efficiently check as correct [15].

VBP uses an authenticated map of key-value pairs maintained in a sparse Merkle tree (SMT). A VBP server maintains one such tree and populates it incrementally, starting from an empty tree. For better performance, the server operates in rounds, collecting updates within a time period and committing them as a batch at the end of the round. All modifications are public for the participants of the process, which enables them to audit the operation of the server if they wish to do so.

The root of the SMT is the trust anchor for the proof holder. It is critical for a verifier to know that only one root value is produced at the end of each round. This guarantees there exists only one version of the tree for all participants. Our solution uses a set of external auditors who digitally sign the roots of the SMT for that purpose.

4.2 Verifiable State Machines

The SMT, by itself, cannot be used to describe a business process. We assume that the allowed states and transitions of each process are defined in some formalism. This could be based on a general standard, such as BPMN, or some custom formalism. For VBP the main requirements are:

- the process description can be deterministically serialized;
- a process instance's state can be deterministically serialized.

VBP tracks the state of each process instance as a key-value pair where the key is a permanent and unique process identifier and the value represents the current state of the process instance. For example, a single shipment from the Supplier to the Customer might be represented in a single leaf. The hash of the process identifier determines the leaf of the SMT for tracking the evolution of the state of that process instance and the hash of the current state is stored in the designated leaf of the SMT.

It is desirable to use meaningful process instance identifiers, as this reduces the risk of participants mistaking a process instance for another. For that, participants need to agree on the naming convention for process instances, in addition to the verification logic for the business process model they are participating in. For example, the counterparties, invoice number, and date of shipment might be

used to derive the process instance identifier. A single SMT can be used to hold multiple instances of the same kind of process, or of different processes.

As a process is executed, participants send hashes of the updated process states to the VBP server. The server will record each update in the corresponding leaf of the SMT. This enables extraction of a proof of state for that process instance relative to the root hash of the SMT. Anyone who holds the process instance and the state proof can verify independently that the process instance is in fact in that state and that other participants must know the same. If a process instance is altered to indicate that the shipment has been delivered to the Customer, the hash value for this process will change. Anyone might see that this has occurred, but without having access to the underlying process instance data that is hashed, it reveals nothing. Thus, the VBP server functions as an independent witness of the process state, but without having to know anything about the transactions or the business rules, in the manner described as the "privacy limited validation" in Sec. 1.

How the process instance moves from one party to another is beyond the scope of VBP. This could be done by any means of transport agreed upon by the participants. In all cases, the integrity of the process instance is guaranteed by the VBP server. Note that the VBP server does not receive any sensitive data to provide that service. Only meta-data on when processes are started and modified is required.

Our current implementation uses a JavaScript based process definition library to describe the process verification rules. Process definitions are modeled as finite state machines (FSM). The same library generates representations of process instances. These are not based on BPMN, although it would be possible to generate these artifacts from a BPMN tool.

4.3 Verifiable Log of Registry Changes

The evolution of process instances can be modeled as a new SMT being constructed by the VBP server for each round. To ensure that all participants have a consistent view of the evolution of a process, we need to prove that the server is not maintaining parallel histories for a process instance and showing different histories to different participants. This general goal can be broken down in following more specific questions:

- How can we prove that a given state for a particular process instance is the current state?
- How can we prove that state was not changed and changed back in secret at some time in the past? (We have termed this the "flip-flop" attack.)
- How can we efficiently traverse the history of state changes of a process instance?

Eijdenberg et al. [6] discuss the properties of verifiable log backed maps (VLBM) where the history of changes to a simple SMT is recorded in a verifiable log for auditing purposes. However, full audit is required to prove correct operation of

a VLBM. Since a full audit would be impractical in large trees, it could allow a malicious operator to perform flip-flop attacks with low probability of detection.

The data structure used in VBP improves upon the VLBM in this respect. Instead of just the hash of the state of a process instance, each leaf of the SMT in VBP stores a record (H, C, T) where:

- H is the hash of the current state;
- C is the counter of state changes since the process instance was created;
- T is the time of the last state change (expressed as the number of the round when that change was recorded in the SMT).

This data structure enables efficient auditing of the VBP server in a manner similar to the server auditing process described in [3]. For efficency, the VBP server applies updates to the SMT in batches. An auditor keeps the current value of the root hash of the SMT as its internal state and uses that to verify the correctness of the updates submitted by the server. The correctness of the batch of updates that transforms the SMT with the root hash R in round N to the SMT with the root hash R' in round $N + 1$ as follows:

1. The auditor sets R^* to R, to start from the SMT state as of round N.
2. For each process state update, the server presents
   - the record $X = (H, C, T)$ of the process instance state as of round N;
   - the hash chain A linking X to R^*;
   - the updated record $X' = (H', C', T')$.
3. The auditor processes each of those updates by
   - checking that A indeed links X to R^*; this establishes the correctness of the initial state of the record;
   - checking that $C' = C+1$ and $T' = N+1$; this establishes the correctness of the update of the metadata in the record; note that the auditor does not check the state change, because it is auditing the behavior of the VBP server and not of the business process participants;
   - updating R^* to the output value of the hash chain A when the record X is replaced with X'; note that the hash chain A is the same as in the first check, which ensures that the server could not have changed any other records with this update.
4. After processing all updates, the auditor checks that $R^* = R'$; this ensures that the server has presented exactly the set of updates that transformed the SMT from the state with the root R to the state with the root R'.
5. If all the checks pass, the auditor signs the new root hash value R' as approved and the server can use it as a reference relative to which proofs can be delivered to clients and also as the starting state for the next batch of updates.

Crucially, the audit protocol allows multiple auditors to work in parallel and independently sign their approvals, thus avoiding the need for a distributed consensus protocol, which would reintroduce some of the limitations of DLT.

For clients, the VBP server provides an interface for querying the (H, C, T) records, together with the associated hash chain proofs. This enables efficient

traversal of records relating to a particular process. The client can query the latest state of the process, then use the time T of the last change to query the previous state at time $T-1$, and so on until the history of the process instance has been traced back to its creation.

The client can then verify that all the state transitions are indeed valid according to the agreed process model. The client can also verify that the counter of state changes increases by one with each change, which ensures that there are no other changes to the process state outside this reported history.

This mode of operation prevents the possibility that processes could be altered briefly, in collusion with the VBP server operator, for the purpose of producing a fake "proof" of an incorrect state, after which the process would be put back to the correct, unaltered state.

Motivations for doing this attack could vary, for example, our Supplier might wish to obtain a proof attesting more widgets had been shipped than really had, for the purpose of fraudulently recognizing revenue early. Since the shipment process state would be corrected moments later by removing the invalid step from the process data, it would be extremely unlikely that anyone would notice.

This type of fraud mainly affects parties who are not participating in the process, but have some outside interest in it, and rely upon its accuracy for some reason. Without a mechanism to prevent it proactively, this type of fraud can only be detected by full audit.

5 Discussion

5.1 Storage requirements

Keeping the SMT with full modification history takes storage space. Each internal node of the SMT contains a hash value of k bits. In the leaf nodes the size of the hash value dominates over the two integers, so for simplicity we can assume equal-sized nodes. As only the hashes of the process states are kept in the SMT, the space requirement does not depend on the size of process states, but only on the number of process instances and their modification rates.

When a full SMT would be naively stored for each round, each process instance would fill one leaf node and cause at most k internal nodes to be populated with non-empty values, for a total of about $N \cdot M \cdot k$ nodes for M processes maintained over N rounds.

However, if only a minority of the processes change in each round, most nodes of the SMT remain the same from one round to the next and those nodes can be shared between the two trees, thus reducing the storage to $N \cdot C \cdot k$, where C is the average number of process instances that change per round.

Since the tree is a sparse one, most paths from a non-empty leaf to the root have many empty siblings. Neither these siblings nor the parents computed from them need to be stored, as each such parent can be re-computed on demand from the only non-empty child. With this optimization, only $\log_2 M$ nodes need to be kept on an average path, for a total of $N \cdot C \cdot \log_2 M$ nodes, each containing k bits, or $k/8$ bytes of data.

The process count is not a constant, however. The append-only nature of the SMT means it is always growing. We can assume constant rate of new process instances to get some useful estimates. For example, with 1 000 clients and 1 000 new process instances per client per year, and an average of 10 state modifications per process instance, we get the rate of about 10 million updates per year and the tree with 1 million leaves by the end of the first year, 2 million leaves by the end of the second year, etc. Some example schedules of storage requirements are listed in Table 1.

Table 1. Storage requirements of a VBP server, depending on the number of clients, number of new process instances per client per year, and the number of state updates over the lifetime of a process instance, assuming a 256-bit hash function.

Clients	Processes	Updates	Year 1	Year 2	Year 3	Year 4	Year 5	Total
1 000	1 000	10	6.4 GB	6.7 GB	6.9 GB	7.0 GB	7.1 GB	34.1 GB
10 000	10 000	100	8.5 TB	8.8 TB	9.0 TB	9.1 TB	9.2 TB	44.7 TB

It is up to the application to decide how long historical state should be stored. A rolling window approach (e.g., store only last year's worth of full proof history) could be applicable in many cases.

5.2 Deployment models

Depending on the business needs, there are multiple ways the VBP service could be deployed. A few examples:

1. Internal organization process auditing: Deploy a VBP server for the organization only. No need for external auditors. Root publication does not have to be global.
2. Coordinate processes within a consortium: Deploy one VBP server for the consortium. Appoint auditors trusted by the consortium members for the VBP server validation and root attestation.
3. Need to prove small number of processes in global scale: Use a global VBP service provider. Service provider handles VBP service auditing and root publication for many clients. Processes are provable independently from the infrastructure of any one organization.
4. Need to prove many processes, some locally, some globally: Use a layered federated architecture, as shown on Fig. 3. Intra-organizationally, short proofs up to the root of the local tree suffice. Cross-organizationally, full proofs up to the root of the global VBP publishing service are needed.

6 Conclusions

We have proposed an alternative to distributed ledgers for adding integrity to multi-party business process automation. This solution uses a "trust, but verify"

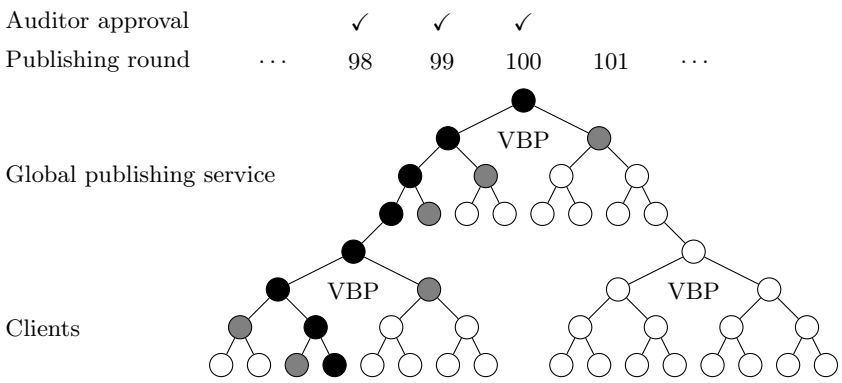

Fig. 3. VBP root publication scheme.

model, which allows us to create a highly scalable solution. In addition, our solution does not require each participant to install new infrastructure. The central piece—the VBP server—could be shared between many participants in a software-as-a-service (SaaS) model.

Our main contribution is adding efficient auditability of modification history to the sparse Merkle tree (SMT) based authenticated data structure, which protects clients against malicious behavior by the service provider, including advanced threats, such as the "flip-flop" attack (cf. Sec. 4.3).

The other contribution is showing how such an authenticated data structure could be used in multi-party business process automation use cases.

7 Future work

A known limitation of VBP is that it can handle only linear processes. Extending the model to support branching, parallel execution, and merging is an interesting avenue of future research.

Another direction would be to develop a root publication service, so it would be usable by other VBP server instances. The layered deployment model proposed in Fig. 3 is conceptual and needs more research.

The solution is not yet integrated into any established BPM systems. This is a possible direction of future practical development effort. As the VBP server is agnostic to what data it contains, multiple such integrations could be supported even by a single instance of the VBP service.

Also, it would be beneficial to implement more real-life use cases with this model, to find shortcomings that could be addressed in the future. Currently, a previous version of VBP is used in one production system for monitoring the business process execution and easing the auditing process of the Certus service [13].

Acknowledgements. This research was partly supported by the EU H2020 project PRIViLEDGE (grant 780477). The authors are also grateful to the anonymous reviewers who pointed out additional related work.

References

1. M. Bauer. Proofs of zero knowledge. https://arxiv.org/abs/cs/0406058, 2004.
2. R. Breu, S. Dustdar, J. Eder, C. Huemer, G. Kappel, J. Köpke, P. Langer, J. Mangler, J. Mendling, G. Neumann, S. Rinderle-Ma, S. Schulte, S. Sobernig, and B. Weber. Towards living inter-organizational processes. In *2013 IEEE 15th Conference on Business Informatics*, pages 363–366. IEEE, 2013.
3. A. Buldas, R. Laanoja, and A. Truu. A blockchain-assisted hash-based signature scheme. In *NordSec 2018*, volume 11252 of *LNCS*, pages 138–153. Springer, 2018.
4. R. Dahlberg, T. Pulls, and R. Peeters. Efficient sparse Merkle trees. In *NordSec 2016*, volume 10014 of *LNCS*, pages 199–215. Springer, 2016.
5. C. Di Ciccio, A. Cecconi, M. Dumas, L. García-Bañuelos, O. López-Pintado, Q. Lu, J. Mendling, A. Ponomarev, A. B. Tran, and I. Weber. Blockchain support for collaborative business processes. *Informatik Spektrum*, 42:182–190, May 2019.
6. A. Eijdenberg, B. Laurie, and A. Cutter. Verifiable data structures. https://www.continusec.com/static/VerifiableDataStructures.pdf, 2015.
7. O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, and A. Ponomarev. Caterpillar: A business process execution engine on the Ethereum blockchain. *Software: Practice and Experience*, 47(7):1162–1193, Jul 2019.
8. J. Mendling, I. Weber, W. Aalst, J. v. Brocke, C. Cabanillas, F. Daniel, S. Debois, C. Di Ciccio, M. Dumas, S. Dustdar, A. Gal, L. García-Bañuelos, G. Governatori, R. Hull, M. La Rosa, H. Leopold, F. Leymann, J. Recker, M. Reichert, and L. Zhu. Blockchains for business process management—challenges and opportunities. *ACM Transactions on Management Information Systems*, 9(1), Feb 2018.
9. R. C. Merkle. *Secrecy, Authentication and Public Key Systems.* PhD thesis, Stanford University, 1979.
10. NIST. Recommendation for key management. SP 800-57, Part 1, Rev. 5, 2020.
11. C. Prybila, S. Schulte, C. Hochreiner, and I. Weber. Runtime verification for business processes utilizing the Bitcoin blockchain. *Future Generation Computer Systems*, 107:816–831, Jun 2020.
12. P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *FSE 2004, Revised Papers*, volume 3017 of *LNCS*, pages 371–388. Springer, 2004.
13. SICPA. Certus. https://www.certusdoc.com [2020-06-13].
14. Stratumn. Proof of process. https://www.proofofprocess.org [2020-05-20].
15. R. Tamassia. Authenticated data structures. In *European Symposium on Algorithms, ESA 2003, Proceedings*, volume 2832 of *LNCS*, pages 2–5. Springer, 2003.
16. A. B. Tran, Q. Lu, and I. Weber. Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management. In *BPM 2018*, volume 2196 of *CEUR Workshop Proceedings*, pages 56–60. CEUR-WS.org, 2018.
17. I. Weber, Q. Lu, A. B. Tran, A. Deshmukh, M. Gorski, and M. Strazds. A platform architecture for multi-tenant blockchain-based systems. In *ICSA 2019*, pages 101–110. IEEE, 2019.